Red Hat OpenStack Platform 15

高可用性デプロイメントと使用方法

Red Hat OpenStack Platform における高可用性のプランニング、デプロイ、および管理

Last Updated: 2022-09-15

Red Hat OpenStack Platform 15 高可用性デプロイメントと使用方法

Red Hat OpenStack Platform における高可用性のプランニング、デプロイ、および管理

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

OpenStack の環境が効率的に稼働する状態を維持するためには、Red Hat OpenStack Platform director を使用して、OpenStack の主要な全サービスにわたって高可用性および負荷分散を提供する構成を構築します。

目次

第1章 高可用性サービス

Red Hat OpenStack Platform (RHOSP) は、高可用性 (HA) を実装するのに必要なサービスを提供するための、さまざまなテクノロジーを採用しています。

サービスの種別

コアコンテナー

コアコンテナーサービスには、**Galera**、**RabbitMQ**、**Redis**、および **HAProxy** が含まれます。これらのサービスはすべてのコントローラノード上で実行され、開始、停止、再起動の各処理に固有の管理と制約が必要です。Pacemaker を使用して、コアコンテナーサービスの起動、管理、およびトラブルシューティングを行います。

注記

RHOSP では、MariaDB Galera Cluster を使用してデータベースのレプリケーションを管理します。

アクティブ/パッシブ

アクティブ/パッシブ のサービスは、1 回に 1 つの Controller ノードでのみ実行され、**openstack-cinder-volume** などのサービスが含まれます。アクティブ/パッシブのサービスを移動するには、Pacemaker を使用して正しい停止/起動シーケンスが実施されるようにする必要があります。

systemd とプレーンコンテナー

systemd およびプレーンコンテナーのサービスは独立したサービスで、サービスの中断に対する耐性があります。したがって、Galera 等の高可用性サービスを再起動した場合、**nova-api** 等の他のサービスを手動で再起動する必要はありません。systemd または Podman を使用して、systemd およびプレーンコンテナーのサービスを直接管理することができます。

director を使用して HA デプロイメントをオーケストレーションする場合、director はテンプレートおよび Puppet モジュールを使用して、すべてのサービスが正しく設定および起動されるようにします。また、HA の問題のトラブルシューティングを行う場合、**podman** コマンドまたは **systemctl** コマンドを使用して、HA フレームワークのサービスと対話する必要があります。

サービスのモード

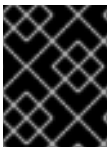
HA サービスは、以下のいずれかのモードで動作することができます。

- **アクティブ/アクティブ**: Pacemaker は同じサービスを複数のコントローラーノードで実行し、HAProxy を使用してトラフィックをノード間に分散するか、または 1 つの IP アドレスにより特定のコントローラーに転送します。HAProxy はラウンドロビンのスケジューリングを使用して、アクティブ/アクティブのサービスにトラフィックを分散する場合もあります。コントローラーノードを追加して、パフォーマンスを向上させることができます。

- **アクティブ/パッシブ**: アクティブ/アクティブモードで実行することのできないサービスは、アクティブ/パッシブモードで実行する必要があります。このモードでは、1 度にアクティブにできるサービスのインスタンスは 1 つだけです。たとえば、HAProxy は stick-table オプションを使用して、受信した Galera データベースの接続要求を 1 つのバックエンドサービスに転送します。このモードは、複数の Galera ノードから同じデータに同時に多数の接続が行われるのを防ぐのに役立ちます。

第2章 デプロイメントの例: COMPUTE および CEPH サービスを持つ高可用性クラスター

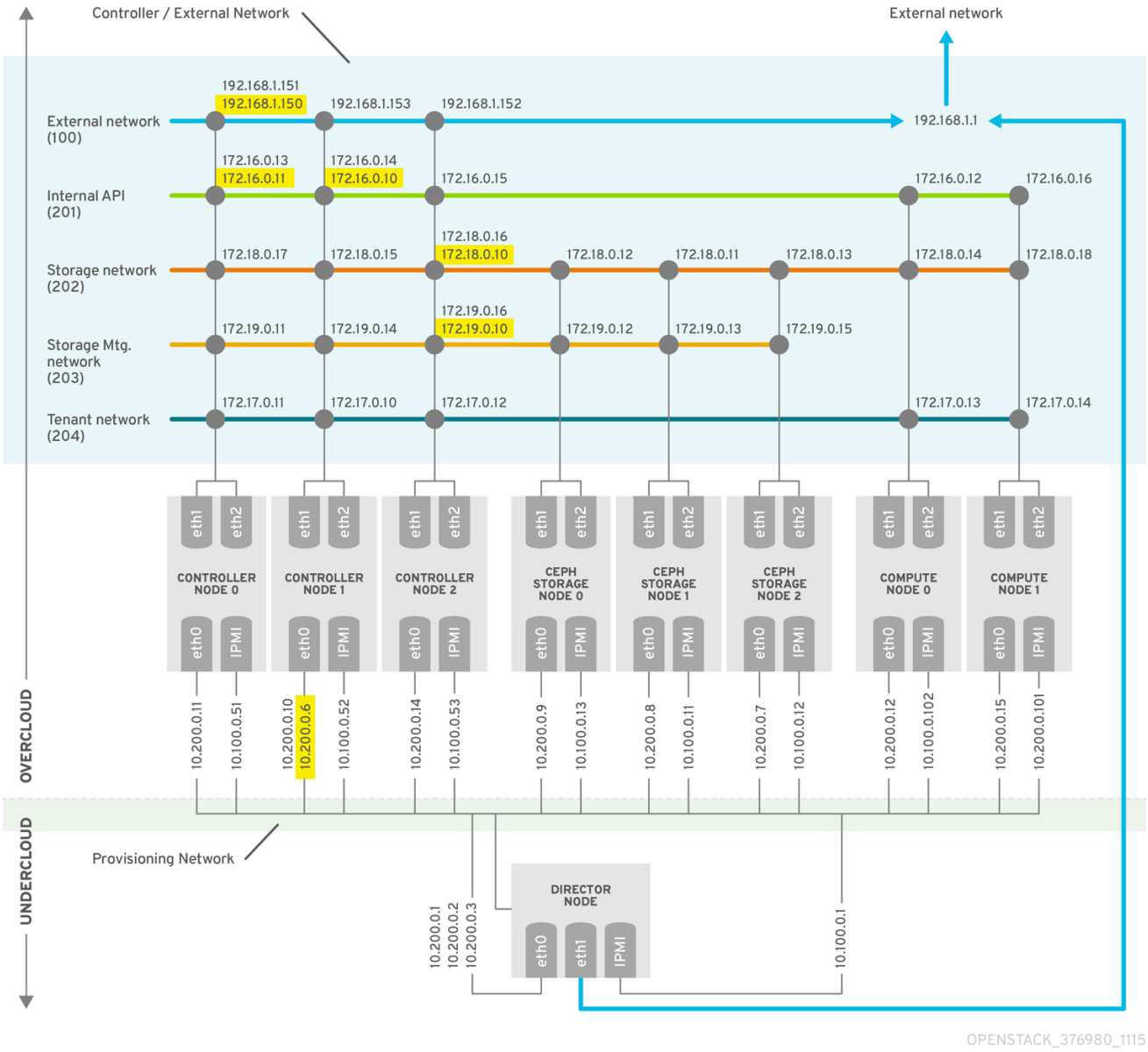
以下のシナリオ例で、OpenStack Compute サービスおよび Red Hat Ceph Storage を持つ高可用性デプロイメントのアーキテクチャー、ハードウェアおよびネットワークの仕様、ならびにアンダークラウドおよびオーバークラウドの設定ファイルについて説明します。



重要

このデプロイメントはテスト環境の参照用として使用することを目的としており、実稼働環境用としてはサポートされません。

図2.1 高可用性デプロイメントアーキテクチャーの例



Red Hat Ceph Storage クラスターのデプロイについての詳しい情報は、『コンテナー化された Red Hat Ceph を持つオーバークラウドのデプロイ』を参照してください。

director を使用した Red Hat OpenStack Platform のデプロイについての詳しい情報は、『director のインストールと使用方法』を参照してください。

2.1. ハードウェアの仕様

以下の表は、デプロイメント例で使用されているハードウェアを示しています。ご自分のテストデプロイメントのニーズに応じて、CPU、メモリー、ストレージ、または NIC を調整することができます。

表2.1 物理コンピューター

コンピューターの台数	目的	CPU の数	メモリー容量	ディスク容量	電源管理	NIC の数
1	アンダークラウドノード	4	6144 MB	40 GB	IPMI	2 (外部 x 1、プロビジョニング x 1) + 1 IPMI
3	コントローラーノード	4	6144 MB	40 GB	IPMI	3 (オーバークラウド上のボンディング x 2、プロビジョニング x 1) + 1 IPMI
3	Ceph Storage ノード	4	6144 MB	40 GB	IPMI	3 (オーバークラウド上のボンディング x 2、プロビジョニング x 1) + 1 IPMI
2	コンピュートノード (必要に応じて追加する)	4	6144 MB	40 GB	IPMI	3 (オーバークラウド上のボンディング x 2、プロビジョニング x 1) + 1 IPMI

ハードウェアの割り当てを計画する際には、以下のガイドラインを確認してください。

コントローラーノード

ストレージ以外のほとんどのサービスは、コントローラノード上で実行される。すべてのサービスは 3 つのノードにわたって複製され、アクティブ/アクティブまたはアクティブ/パッシブのサービスとして設定される。HA 環境には、最低でも 3 つのノードが必要である。

Red Hat Ceph Storage ノード

ストレージサービスはこれらのノード上で実行され、コンピュートノードに Red Hat Ceph Storage 領域プールを提供する。最低でも 3 つのノードが必要である。

コンピュートノード

仮想マシン (VM) インスタンスは、コンピュートノード上で実行される。能力の要件ならびに移行およびリブート操作に必要な数だけ、コンピュートノードをデプロイすることができる。仮想マシンがストレージノード、他のコンピュートノード上の仮想マシン、およびパブリックネットワークに

アクセスできるようにするため、コンピュートノードをストレージネットワークおよびテナントネットワークに接続する必要がある。

STONITH

高可用性オーバークラウドの Pacemaker クラスターの一部である各ノードには、STONITH デバイスを設定する必要があります。STONITH を使用しない高可用性オーバークラウドのデプロイはサポートの対象外です。STONITH および Pacemaker の詳細は、「Fencing in a Red Hat High Availability Cluster」および「Support Policies for RHEL High Availability Clusters - General Requirements for Fencing/STONITH」を参照してください。

2.2. ネットワークの仕様

以下の表は、デプロイメント例で使用されているネットワーク構成を示しています。

表2.2 物理ネットワークおよび仮想ネットワーク

物理 NIC	目的	VLAN	説明
eth0	プロビジョニングネットワーク (アンダークラウド)	該当なし	すべてのノードを director (アンダークラウド) から管理します。
eth1 および eth2	コントローラー/外部 (オーバークラウド)	該当なし	ボンディングされた NIC および VLAN の組み合わせ
	外部ネットワーク	VLAN 100	外部環境からテナントネットワーク、内部 API、および OpenStack Horizon Dashboard へのアクセスを許可します。
	内部 API	VLAN 201	コンピュートノードとコントローラーノード間の内部 API へのアクセスを提供します。
	ストレージアクセス	VLAN 202	コンピュートノードをストレージメディアに接続します。
	ストレージ管理	VLAN 203	ストレージメディアを管理します。
	テナントネットワーク	VLAN 204	RHOSP にテナントネットワークサービスを提供します。

ネットワーク構成に加えて、以下のコンポーネントをデプロイする必要があります。

プロビジョニングネットワーク用のスイッチ

- このスイッチは、アンダークラウドをオーバークラウド内のすべての物理コンピューターに接続できる必要があります。
- このスイッチに接続されている各オーバークラウドノードの NIC は、アンダークラウドから PXE ブートできなければなりません。
- **portfast** パラメータを有効にする必要があります。

コントローラー/外部ネットワーク用のスイッチ

- このスイッチは、デプロイメント内の他の VLAN の VLAN タグ付けを行うように設定する必要があります。
- VLAN 100 トラフィックのみを外部ネットワークに許可します。

2.3. アンダークラウドの設定ファイル

デプロイメント例では、以下のアンダークラウド設定ファイルが使用されます。

instackenv.json

```
{
    "nodes": [
      {
        "pm_password": "testpass",
        "memory": "6144",
        "pm_addr": "10.100.0.11",
        "mac": [
          "2c:c2:60:3b:b3:94"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "1",
        "pm_user": "admin"
      },
      {
        "pm_password": "testpass",
        "memory": "6144",
        "pm_addr": "10.100.0.12",
        "mac": [
          "2c:c2:60:51:b7:fb"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "1",
        "pm_user": "admin"
      },
      {
        "pm_password": "testpass",
        "memory": "6144",
        "pm_addr": "10.100.0.13",
        "mac": [
          "2c:c2:60:76:ce:a5"
```

```
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.51",
    "mac": [
      "2c:c2:60:08:b1:e2"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.52",
    "mac": [
      "2c:c2:60:20:a1:9e"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.53",
    "mac": [
      "2c:c2:60:58:10:33"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "6144",
    "pm_addr": "10.100.0.101",
    "mac": [
      "2c:c2:60:31:a9:55"
    ],
    "pm_type": "pxe_ipmitool",
    "disk": "40",
    "arch": "x86_64",
```

```
        "cpu": "2",
        "pm_user": "admin"
      },
      {
        "pm_password": "testpass",
        "memory": "6144",
        "pm_addr": "10.100.0.102",
        "mac": [
          "2c:c2:60:0d:e7:d1"
        ],
        "pm_type": "pxe_ipmitool",
        "disk": "40",
        "arch": "x86_64",
        "cpu": "2",
        "pm_user": "admin"
      }
    ],
    "overcloud": {"password": "7adbbbeedc5b7a07ba1917e1b3b228334f9a2d4e",
    "endpoint": "http://192.168.1.150:5000/v2.0/"
              }
}
```

undercloud.conf

```
[DEFAULT]
image_path = /home/stack/images
local_ip = 10.200.0.1/24
undercloud_public_vip = 10.200.0.2
undercloud_admin_vip = 10.200.0.3
undercloud_service_certificate = /etc/pki/instack-certs/undercloud.pem
local_interface = eth0
masquerade_network = 10.200.0.0/24
dhcp_start = 10.200.0.5
dhcp_end = 10.200.0.24
network_cidr = 10.200.0.0/24
network_gateway = 10.200.0.1
#discovery_interface = br-ctlplane
discovery_iprange = 10.200.0.150,10.200.0.200
discovery_runbench = 1
undercloud_admin_password = testpass
...
```

network-environment.yaml

```
resource_registry:
  OS::TripleO::BlockStorage::Net::SoftwareConfig: /home/stack/templates/nic-configs/cinder-
storage.yaml
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-configs/controller.yaml
  OS::TripleO::ObjectStorage::Net::SoftwareConfig: /home/stack/templates/nic-configs/swift-
storage.yaml
  OS::TripleO::CephStorage::Net::SoftwareConfig: /home/stack/templates/nic-configs/ceph-
storage.yaml

parameter_defaults:
```

```
  InternalApiNetCidr: 172.16.0.0/24
  TenantNetCidr: 172.17.0.0/24
  StorageNetCidr: 172.18.0.0/24
  StorageMgmtNetCidr: 172.19.0.0/24
  ExternalNetCidr: 192.168.1.0/24
  InternalApiAllocationPools: [{'start': '172.16.0.10', 'end': '172.16.0.200'}]
  TenantAllocationPools: [{'start': '172.17.0.10', 'end': '172.17.0.200'}]
  StorageAllocationPools: [{'start': '172.18.0.10', 'end': '172.18.0.200'}]
  StorageMgmtAllocationPools: [{'start': '172.19.0.10', 'end': '172.19.0.200'}]
  # Leave room for floating IPs in the External allocation pool
  ExternalAllocationPools: [{'start': '192.168.1.150', 'end': '192.168.1.199'}]
  InternalApiNetworkVlanID: 201
  StorageNetworkVlanID: 202
  StorageMgmtNetworkVlanID: 203
  TenantNetworkVlanID: 204
  ExternalNetworkVlanID: 100
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 192.168.1.1
  # Set to "br-ex" if using floating IPs on native VLAN on bridge br-ex
  NeutronExternalNetworkBridge: """"
  # Customize bonding options if required
  BondInterfaceOvsOptions:
    "bond_mode=active-backup lacp=off other_config:bond-miimon-interval=100"
```

2.4. オーバークラウドの設定ファイル

デプロイメント例では、以下のオーバークラウド設定ファイルが使用されます。

/var/lib/config-data/haproxy/etc/haproxy/haproxy.cfg (コントローラーノード)

このファイルは、HAProxy が管理するサービスを特定します。これには、HAProxy がモニタリングするサービスの設定が含まれます。このファイルは、すべてのコントローラーノードで同じ内容です。

```
# This file is managed by Puppet
global
  daemon
  group  haproxy
  log  /dev/log local0
  maxconn  20480
  pidfile  /var/run/haproxy.pid
  ssl-default-bind-ciphers
!SSLv2:kEECDH:kRSA:kEDH:kPSK:+3DES:!aNULL:!eNULL:!MD5:!EXP:!RC4:!SEED:!IDEA:!DES
  ssl-default-bind-options  no-sslv3
  stats  socket /var/lib/haproxy/stats mode 600 level user
  stats  timeout 2m
  user  haproxy

defaults
  log  global
  maxconn  4096
  mode  tcp
  retries  3
  timeout  http-request 10s
  timeout  queue 2m
  timeout  connect 10s
```

```
  timeout  client 2m
  timeout  server 2m
  timeout  check 10s

listen aodh
  bind 192.168.1.150:8042 transparent
  bind 172.16.0.10:8042 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8042 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8042 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8042 check fall 5 inter 2000 rise 2

listen cinder
  bind 192.168.1.150:8776 transparent
  bind 172.16.0.10:8776 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8776 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8776 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8776 check fall 5 inter 2000 rise 2

listen glance_api
  bind 192.168.1.150:9292 transparent
  bind 172.18.0.10:9292 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk GET /healthcheck
  server overcloud-controller-0.internalapi.localdomain 172.18.0.17:9292 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.18.0.15:9292 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.18.0.16:9292 check fall 5 inter 2000 rise 2

listen gnocchi
  bind 192.168.1.150:8041 transparent
  bind 172.16.0.10:8041 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8041 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8041 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8041 check fall 5 inter 2000 rise 2

listen haproxy.stats
  bind 10.200.0.6:1993 transparent
  mode http
  stats enable
  stats uri /
  stats auth admin:PnDD32EzdVCf73CpjHhFGHZdV

listen heat_api
```

```
  bind 192.168.1.150:8004 transparent
  bind 172.16.0.10:8004 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  timeout client 10m
  timeout server 10m
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8004 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8004 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8004 check fall 5 inter 2000 rise 2

listen heat_cfn
  bind 192.168.1.150:8000 transparent
  bind 172.16.0.10:8000 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  timeout client 10m
  timeout server 10m
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8000 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8000 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8000 check fall 5 inter 2000 rise 2

listen horizon
  bind 192.168.1.150:80 transparent
  bind 172.16.0.10:80 transparent
  mode http
  cookie SERVERID insert indirect nocache
  option forwardfor
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:80 check cookie overcloud-
controller-0 fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:80 check cookie overcloud-
controller-0 fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:80 check cookie overcloud-
controller-0 fall 5 inter 2000 rise 2

listen keystone_admin
  bind 192.168.24.15:35357 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk GET /v3
  server overcloud-controller-0.ctlplane.localdomain 192.168.24.9:35357 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.ctlplane.localdomain 192.168.24.8:35357 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.ctlplane.localdomain 192.168.24.18:35357 check fall 5 inter 2000 rise
2

listen keystone_public
  bind 192.168.1.150:5000 transparent
  bind 172.16.0.10:5000 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
```

```
  option httpchk GET /v3
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:5000 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:5000 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:5000 check fall 5 inter 2000 rise 2

listen mysql
  bind 172.16.0.10:3306 transparent
  option tcpka
  option httpchk
  stick on dst
  stick-table type ip size 1000
  timeout client 90m
  timeout server 90m
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:3306 backup check inter 1s on-
marked-down shutdown-sessions port 9200
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:3306 backup check inter 1s on-
marked-down shutdown-sessions port 9200
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:3306 backup check inter 1s on-
marked-down shutdown-sessions port 9200

listen neutron
  bind 192.168.1.150:9696 transparent
  bind 172.16.0.10:9696 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:9696 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:9696 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:9696 check fall 5 inter 2000 rise 2

listen nova_metadata
  bind 172.16.0.10:8775 transparent
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8775 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8775 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8775 check fall 5 inter 2000 rise 2

listen nova_novncproxy
  bind 192.168.1.150:6080 transparent
  bind 172.16.0.10:6080 transparent
  balance source
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option tcpka
  timeout tunnel 1h
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:6080 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:6080 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:6080 check fall 5 inter 2000 rise 2

listen nova_osapi
  bind 192.168.1.150:8774 transparent
  bind 172.16.0.10:8774 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
```

```
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8774 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8774 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8774 check fall 5 inter 2000 rise 2

listen nova_placement
  bind 192.168.1.150:8778 transparent
  bind 172.16.0.10:8778 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8778 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8778 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8778 check fall 5 inter 2000 rise 2

listen panko
  bind 192.168.1.150:8977 transparent
  bind 172.16.0.10:8977 transparent
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8977 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8977 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8977 check fall 5 inter 2000 rise 2

listen redis
  bind 172.16.0.13:6379 transparent
  balance first
  option tcp-check
  tcp-check send AUTH\ V2EgUh2pvkr8VzU6yuE4XHsr9\r\n
  tcp-check send PING\r\n
  tcp-check expect string +PONG
  tcp-check send info\ replication\r\n
  tcp-check expect string role:master
  tcp-check send QUIT\r\n
  tcp-check expect string +OK
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:6379 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:6379 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:6379 check fall 5 inter 2000 rise 2

listen swift_proxy_server
  bind 192.168.1.150:8080 transparent
  bind 172.18.0.10:8080 transparent
  option httpchk GET /healthcheck
  timeout client 2m
  timeout server 2m
  server overcloud-controller-0.storage.localdomain 172.18.0.17:8080 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.storage.localdomain 172.18.0.15:8080 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.storage.localdomain 172.18.0.16:8080 check fall 5 inter 2000 rise 2
```

/etc/corosync/corosync.conf ファイル (コントローラーノード)

このファイルは、クラスターのインフラストラクチャーを定義し、すべてのコントローラーノード上で利用することができます。

```
totem {
 version: 2
 cluster_name: tripleo_cluster
 transport: udpu
 token: 10000
}


nodelist {
   node {
      ring0_addr: overcloud-controller-0
      nodeid: 1
   }


   node {
      ring0_addr: overcloud-controller-1
      nodeid: 2
   }


   node {
      ring0_addr: overcloud-controller-2
      nodeid: 3
   }
}

quorum {
   provider: corosync_votequorum
}

logging {
    to_logfile: yes
   logfile: /var/log/cluster/corosync.log
   to_syslog: yes
}
```

/etc/ceph/ceph.conf (Ceph ノード)

このファイルには、Ceph の高可用性設定が記載されています。これには、モニタリングするホストのホスト名、IP アドレスが含まれます。

```
[global]
osd_pool_default_pgp_num = 128
osd_pool_default_min_size = 1
auth_service_required = cephx
mon_initial_members = overcloud-controller-0,overcloud-controller-1,overcloud-controller-2
fsid = 8c835acc-6838-11e5-bb96-2cc260178a92
cluster_network = 172.19.0.11/24
auth_supported = cephx
auth_cluster_required = cephx
mon_host = 172.18.0.17,172.18.0.15,172.18.0.16
auth_client_required = cephx
osd_pool_default_size = 3
osd_pool_default_pg_num = 128
public_network = 172.18.0.17/24
```

-

第3章 高可用性環境へのアクセス

アンダークラウドから特定の HAノードの詳細にアクセスすることができます。

手順

1. 動作中の HA 環境で、アンダークラウドにログインします。
2. **stack** ユーザーに切り替えます。

```
# sudo su - stack
```

3. アンダークラウドノードの詳細にアクセスするには、アンダークラウドからそのノードの IP アドレスを取得します。

```
(undercloud) $ source ~/stackrc
(undercloud) $ openstack server list
 +-------+-----------------------+---+---------------------+---+
 | ID    | Name                  |...| Networks            |...|
 +-------+-----------------------+---+---------------------+---+
 | d1... | overcloud-controller-0 |...| ctlplane=*10.200.0.11* |...|
 ...
```

4. オーバークラウドノードのいずれかにログインするには、以下のコマンドを実行します。

```
(undercloud) $ source ~/overcloudrc
(overcloud) $ ssh [NODE_NAME]@[NODE_IP]
```

名前および IP アドレスをデプロイメントの実際の値に置き換えます。

第4章 PACEMAKER を使用した高可用性サービスの管理

Pacemaker サービスは、Galera、RabbitMQ、Redis、および HAProxy 等のコアコンテナーのサービスおよびアクティブパッシブのサービスを管理します。Pacemaker を使用して、管理対象サービス、仮想 IP アドレス、電源管理、およびフェンシングについての一般的な情報を表示および管理します。

Red Hat Enterprise Linux の Pacemaker の詳細については、Red Hat Enterprise Linux ドキュメントの『高可用性クラスターの設定および管理』を参照してください。

4.1. リソースバンドルとコンテナー

Pacemaker は Red Hat OpenStack Platform (RHOSP) のサービスを **バンドルセットリソース** (**バンドル**) として管理します。これらのサービスのほとんどはアクティブ/アクティブのサービスで、それぞれのコントローラーノード上で同じように起動し常に動作しています。

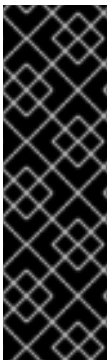
ペースメーカーは以下のリソース種別を管理します。

バンドル

バンドルリソースはすべてのコントローラノードで同じコンテナーを設定および複製し、必要なストレージパスをコンテナーディレクトリーにマッピングし、リソース自体に関連する特定の属性を設定します。

Container

コンテナーは、HAProxy のような単純な **systemd** サービスから、異なるノード上のサービスの状態を制御および設定する特定のリソースエージェントを必要とする Galera のような複雑なサービスまで、さまざまな種類のリソースを実行することができます。



重要

- バンドルまたはコンテナーを管理するのに、**podman** または **systemctl** を使用することはできません。これらのコマンドを使用してサービスのステータスを確認することはできますが、これらのサービスに対してアクションを実行するには Pacemaker を使用する必要があります。
- Pacemaker が制御する Podman コンテナーでは、Podman により **RestartPolicy** が **no** に設定されます。これは、Podman ではなく Pacemaker がコンテナーの起動と停止のアクションを制御するようにするためです。

簡易バンドルセットリソース (簡易バンドル)

簡易バンドルセットリソース (**簡易バンドル**) はコンテナーのセットで、それぞれのコンテナーには全コントローラーノードにわたってデプロイする同じ Pacemaker サービスが含まれます。

以下の例は、**pcs status** コマンドで出力される簡易バンドルの一覧を示しています。

```
Podman container set: haproxy-bundle [192.168.24.1:8787/rhosp-rhel8/openstack-haproxy:pcmklatest]
  haproxy-bundle-podman-0    (ocf::heartbeat:podman):    Started overcloud-controller-0
  haproxy-bundle-podman-1    (ocf::heartbeat:podman):    Started overcloud-controller-1
  haproxy-bundle-podman-2    (ocf::heartbeat:podman):    Started overcloud-controller-2
```

各バンドルでは、以下の情報を確認することができます。

- Pacemaker がサービスに割り当てる名前

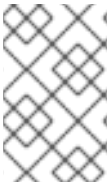
- バンドルに関連付けられたコンテナーへの参照
- 異なるコントローラーノードで実行中のレプリカの一覧およびステータス

以下の例は、**haproxy-bundle** 簡易バンドルの設定を示しています。

```
$ sudo pcs resource show haproxy-bundle
Bundle: haproxy-bundle
 Podman: image=192.168.24.1:8787/rhosp-rhel8/openstack-haproxy:pcmklatest network=host
options="--user=root --log-driver=journald -e KOLLA_CONFIG_STRATEGY=COPY_ALWAYS"
replicas=3 run-command="/bin/bash /usr/local/bin/kolla_start"
 Storage Mapping:
  options=ro source-dir=/var/lib/kolla/config_files/haproxy.json target-
dir=/var/lib/kolla/config_files/config.json (haproxy-cfg-files)
  options=ro source-dir=/var/lib/config-data/puppet-generated/haproxy/ target-
dir=/var/lib/kolla/config_files/src (haproxy-cfg-data)
  options=ro source-dir=/etc/hosts target-dir=/etc/hosts (haproxy-hosts)
  options=ro source-dir=/etc/localtime target-dir=/etc/localtime (haproxy-localtime)
  options=ro source-dir=/etc/pki/ca-trust/extracted target-dir=/etc/pki/ca-trust/extracted (haproxy-pki-
extracted)
  options=ro source-dir=/etc/pki/tls/certs/ca-bundle.crt target-dir=/etc/pki/tls/certs/ca-bundle.crt
(haproxy-pki-ca-bundle-crt)
  options=ro source-dir=/etc/pki/tls/certs/ca-bundle.trust.crt target-dir=/etc/pki/tls/certs/ca-
bundle.trust.crt (haproxy-pki-ca-bundle-trust-crt)
  options=ro source-dir=/etc/pki/tls/cert.pem target-dir=/etc/pki/tls/cert.pem (haproxy-pki-cert)
  options=rw source-dir=/dev/log target-dir=/dev/log (haproxy-dev-log)
```

この例では、バンドル内のコンテナーについて以下の情報を示しています。

- **image**: コンテナーによって使用されるイメージ。アンダークラウドのローカルレジストリーを参照します。
- **network**: コンテナーのネットワーク種別。この例では **"host"** です。
- **options**: コンテナーの特定のオプション
- **replicas**: クラスター内で実行する必要のあるコンテナーのコピーの数を示します。各バンドルには 3 つのコンテナーが含まれ、それぞれが各コントローラーノードに対応します。
- **run-command**: コンテナーの起動に使用するシステムコマンド
- Storage Mapping: 各ホスト上のローカルパスからコンテナーへのマッピング。ホストから haproxy 設定を確認するには、**/etc/haproxy/haproxy.cfg** ファイルの代わりに **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** ファイルを開きます。



注記

HAProxy は、トラフィックの負荷を選択したサービスに分散することによって高可用性サービスを提供しますが、ここでは HAProxy を Pacemaker のバンドルサービスとして管理することによって HAProxy を高可用性サービスに設定します。

複合バンドルセットリソース (複合バンドル)

複合バンドルセットリソース (**複合バンドル**) は、簡易バンドルに含まれる基本的なコンテナーの設定に加えて、リソース設定を指定する Pacemaker サービスです。

この設定は、実行するコントローラノードに応じて異なる状態を取ることができるサービスである **multi-state** のリソースを管理するのに必要です。

以下の例には、**pcs status** コマンドで出力される複合バンドルの一覧を示しています。

```
Podman container set: rabbitmq-bundle [192.168.24.1:8787/rhosp-rhel8/openstack-
rabbitmq:pcmklatest]
  rabbitmq-bundle-0    (ocf::heartbeat:rabbitmq-cluster):      Started overcloud-controller-0
  rabbitmq-bundle-1    (ocf::heartbeat:rabbitmq-cluster):      Started overcloud-controller-1
  rabbitmq-bundle-2    (ocf::heartbeat:rabbitmq-cluster):      Started overcloud-controller-2
Podman container set: galera-bundle [192.168.24.1:8787/rhosp-rhel8/openstack-mariadb:pcmklatest]
  galera-bundle-0      (ocf::heartbeat:galera):        Master overcloud-controller-0
  galera-bundle-1      (ocf::heartbeat:galera):        Master overcloud-controller-1
  galera-bundle-2      (ocf::heartbeat:galera):        Master overcloud-controller-2
Podman container set: redis-bundle [192.168.24.1:8787/rhosp-rhel8/openstack-redis:pcmklatest]
  redis-bundle-0       (ocf::heartbeat:redis): Master overcloud-controller-0
  redis-bundle-1       (ocf::heartbeat:redis): Slave overcloud-controller-1
  redis-bundle-2       (ocf::heartbeat:redis): Slave overcloud-controller-2
```

この出力は、各複合バンドルについての以下の情報を示しています。

- RabbitMQ: 簡易バンドルと同様に、3 つすべてのコントローラーノードが、サービスのスタンドアロンインスタンスを実行している。
- Galera: 3 つすべてのコントローラーノードが、同じ制約下で Galera マスターとして動作している。
- Redis: **overcloud-controller-0** コンテナーはマスターとして動作し、一方、他の 2 つのコントローラーノードはスレーブとして動作している。それぞれのコンテナー種別は、異なる制約下で動作する可能性があります。

以下の例は、**galera-bundle** 複合バンドルの設定を示しています。

```
[...]
Bundle: galera-bundle
 Podman: image=192.168.24.1:8787/rhosp-rhel8/openstack-mariadb:pcmklatest masters=3
network=host options="--user=root --log-driver=journald -e
KOLLA_CONFIG_STRATEGY=COPY_ALWAYS" replicas=3 run-command="/bin/bash
/usr/local/bin/kolla_start"
 Network: control-port=3123
 Storage Mapping:
  options=ro source-dir=/var/lib/kolla/config_files/mysql.json target-
dir=/var/lib/kolla/config_files/config.json (mysql-cfg-files)
  options=ro source-dir=/var/lib/config-data/puppet-generated/mysql/ target-
dir=/var/lib/kolla/config_files/src (mysql-cfg-data)
  options=ro source-dir=/etc/hosts target-dir=/etc/hosts (mysql-hosts)
  options=ro source-dir=/etc/localtime target-dir=/etc/localtime (mysql-localtime)
  options=rw source-dir=/var/lib/mysql target-dir=/var/lib/mysql (mysql-lib)
  options=rw source-dir=/var/log/mariadb target-dir=/var/log/mariadb (mysql-log-mariadb)
  options=rw source-dir=/dev/log target-dir=/dev/log (mysql-dev-log)
 Resource: galera (class=ocf provider=heartbeat type=galera)
  Attributes: additional_parameters=--open-files-limit=16384 cluster_host_map=overcloud-controller-
0:overcloud-controller-0.internalapi.localdomain;overcloud-controller-1:overcloud-controller-
1.internalapi.localdomain;overcloud-controller-2:overcloud-controller-2.internalapi.localdomain
enable_creation=true wsrep_cluster_address=gcomm://overcloud-controller-
0.internalapi.localdomain,overcloud-controller-1.internalapi.localdomain,overcloud-controller-
```

```
2.internalapi.localdomain
  Meta Attrs: container-attribute-target=host master-max=3 ordered=true
  Operations: demote interval=0s timeout=120 (galera-demote-interval-0s)
              monitor interval=20 timeout=30 (galera-monitor-interval-20)
              monitor interval=10 role=Master timeout=30 (galera-monitor-interval-10)
              monitor interval=30 role=Slave timeout=30 (galera-monitor-interval-30)
              promote interval=0s on-fail=block timeout=300s (galera-promote-interval-0s)
              start interval=0s timeout=120 (galera-start-interval-0s)
              stop interval=0s timeout=120 (galera-stop-interval-0s)
[...]
```

この出力は、簡易バンドルとは異なり、**galera-bundle** リソースには、multi-state リソースのあらゆる側面を決定する明示的なリソース設定が含まれていることを示しています。

注記

また、サービスは同時に複数のコントローラーノードで実行される可能性がありますが、コントローラーノード自体は、これらのサービスにアクセスするのに必要な IP アドレスをリッスンしていない場合もあります。サービスの IP アドレスを確認する方法については、「仮想 IP アドレスの表示」を参照してください。

4.2. PACEMAKER の全般的な情報の表示

Pacemaker の全般的な情報を表示するには、**pcs status** コマンドを使用します。

手順

1. 任意のコントローラーノードに **heat-admin** ユーザーとしてログインします。

```
$ ssh heat-admin@overcloud-controller-0
```

2. **pcs status** コマンドを実行します。

```
[heat-admin@overcloud-controller-0 ~]  $ sudo pcs status
```

出力例:

```
Cluster name: tripleo_cluster
Stack: corosync
Current DC: overcloud-controller-1 (version 2.0.1-4.el8-0eb7991564) - partition with quorum

Last updated: Thu Feb  8 14:29:21 2018
Last change: Sat Feb  3 11:37:17 2018 by root via cibadmin on overcloud-controller-2

12 nodes configured
37 resources configured

Online: [ overcloud-controller-0 overcloud-controller-1 overcloud-controller-2 ]
GuestOnline: [ galera-bundle-0@overcloud-controller-0 galera-bundle-1@overcloud-
controller-1 galera-bundle-2@overcloud-controller-2 rabbitmq-bundle-0@overcloud-
controller-0 rabbitmq-bundle-1@overcloud-controller-1 rabbitmq-bundle-2@overcloud-
controller-2 redis-bundle-0@overcloud-controller-0 redis-bundle-1@overcloud-controller-1
redis-bundle-2@overcloud-controller-2 ]
```

```
Full list of resources:
[...]
```

この出力の主要なセクションでは、クラスターに関する以下の情報が表示されます。

- **Cluster name**: クラスターの名前。
- **[NUM] nodes configured**: クラスターを構成するノードの数
- **[NUM] resources configured**: クラスターに設定されているリソースの数
- **Online**: 現在オンライン状態の Controller ノードの名前
- **GuestOnline**: 現在オンライン状態のゲストノードの名前。各ゲストノードは、複合バンドルセットのリソースで構成されています。バンドルセットの詳細については、「リソースバンドルとコンテナー」を参照してください。

4.3. バンドルのステータスの確認

アンダークラウドノードからバンドルのステータスを確認することも、コントローラノードのいずれかにログインして直接バンドルのステータスを確認することもできます。

アンダークラウドノードからのバンドルステータスの確認

次のコマンドを実行します。

```
$ sudo podman exec -it haproxy-bundle-podman-0 ps -efww | grep haproxy*
```

出力例:

```
root           7       1  0 06:08 ?        00:00:00 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
haproxy       11       7  0 06:08 ?        00:00:17 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
```

この出力は、コンテナー内で **haproxy** プロセスが実行されていることを示しています。

コントローラーノードからのバンドルステータスの確認

コントローラーノードにログインし、以下のコマンドを実行します。

```
$ ps -ef | grep haproxy*
```

出力例:

```
root       17774   17729  0 06:08 ?        00:00:00 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
42454      17819   17774  0 06:08 ?        00:00:21 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
root      288508  237714  0 07:04 pts/0    00:00:00 grep --color=auto haproxy*
[root@controller-0 ~]# ps -ef | grep -e 17774 -e 17819
root       17774   17729  0 06:08 ?        00:00:00 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
42454      17819   17774  0 06:08 ?        00:00:22 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
root      301950  237714  0 07:07 pts/0    00:00:00 grep --color=auto -e 17774 -e 17819
```

4.4. 仮想 IP アドレスの表示

各 IPaddr2 リソースは、クライアントがサービスへのアクセスを要求するために使用する仮想 IP アドレスを設定します。その IP アドレスが割り当てられたコントローラーノードで異常が発生すると、IPaddr2 リソースは IP アドレスを別のコントローラーノードに再割り当てします。

すべての仮想 IP アドレスの表示

--full オプションを指定して **pcs resource show** コマンドを実行し、**VirtualIP** タイプを使用するすべてのリソースを表示します。

```
$ sudo pcs resource show --full
```

以下の出力例では、特定の仮想 IP アドレスをリッスンするように現在設定されている各コントローラーノードを確認することができます。

```
ip-10.200.0.6 (ocf::heartbeat:IPaddr2): Started overcloud-controller-1
ip-192.168.1.150 (ocf::heartbeat:IPaddr2): Started overcloud-controller-0
ip-172.16.0.10 (ocf::heartbeat:IPaddr2): Started overcloud-controller-1
ip-172.16.0.11 (ocf::heartbeat:IPaddr2): Started overcloud-controller-0
ip-172.18.0.10 (ocf::heartbeat:IPaddr2): Started overcloud-controller-2
ip-172.19.0.10 (ocf::heartbeat:IPaddr2): Started overcloud-controller-2
```

初期状態では、各 IP アドレスは特定のコントローラーノードに接続されます。たとえば、**192.168.1.150** は **overcloud-controller-0** で開始されます。ただし、そのコントローラーノードで異常が発生すると、IP アドレスはクラスター内の他のコントローラーノードに再割り当てされます。

以下の表には、この出力例の IP アドレスと、各 IP アドレスの初期の割り当てをまとめています。

表4.1 IP アドレスの説明と割り当て元

IP アドレス	説明	割り当て元
192.168.1.150	パブリック IP アドレス	**network-environment.yaml** ファイルの **ExternalAllocationPools** 属性
10.200.0.6	コントローラーの仮想 IP アドレス	**dhcp_start** および **dhcp_end** の範囲の部分は、**undercloud.conf** ファイルで **10.200.0.5-10.200.0.24** に設定されます。
172.16.0.10	コントローラーノード上の OpenStack API サービスへのアクセスを提供します。	**network-environment.yaml** ファイルの **InternalApiAllocationPools**
172.18.0.10	Glance API および Swift プロキシーのサービスへのアクセスを提供するストレージの仮想 IP アドレス	**network-environment.yaml** ファイルの **StorageAllocationPools** 属性
172.16.0.11	コントローラーノード上の Radis サービスへのアクセスを提供します。	**network-environment.yaml** ファイルの **InternalApiAllocationPools**

IP アドレス	説明	割り当て元
172.19.0.10	ストレージ管理へのアクセスを提供します。	**network-environment.yaml** ファイルの **StorageMgmtAlloctionPools**

特定 IP アドレスの表示

pcs resource show コマンドを実行します。

```
$ sudo pcs resource show ip-192.168.1.150
```

出力例:

```
Resource: ip-192.168.1.150 (class=ocf provider=heartbeat type=IPaddr2)
 Attributes: ip=192.168.1.150 cidr_netmask=32
 Operations: start interval=0s timeout=20s (ip-192.168.1.150-start-timeout-20s)
             stop interval=0s timeout=20s (ip-192.168.1.150-stop-timeout-20s)
             monitor interval=10s timeout=20s (ip-192.168.1.150-monitor-interval-10s)
```

特定 IP アドレスのネットワーク情報の表示

1. 表示する IP アドレスが割り当てられているコントローラーノードにログインします。
2. **ip addr show** コマンドを実行して、ネットワークインターフェース情報を表示します。

   ```
   $ ip addr show vlan100
   ```

   出力例:

   ```
    9: vlan100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
   UNKNOWN
       link/ether be:ab:aa:37:34:e7 brd ff:ff:ff:ff:ff:ff
       inet *192.168.1.151/24* brd 192.168.1.255 scope global vlan100
          valid_lft forever preferred_lft forever
       inet *192.168.1.150/32* brd 192.168.1.255 scope global vlan100
          valid_lft forever preferred_lft forever
   ```

3. **netstat** コマンドを実行して、IP アドレスをリッスンするすべてのプロセスを表示します。

   ```
   $ sudo netstat -tupln | grep "192.168.1.150.haproxy"
   ```

   出力例:

   ```
   tcp        0      0 192.168.1.150:8778      0.0.0.0:*               LISTEN      61029/haproxy
   tcp        0      0 192.168.1.150:8042      0.0.0.0:*               LISTEN      61029/haproxy
   tcp        0      0 192.168.1.150:9292      0.0.0.0:*               LISTEN      61029/haproxy
   tcp        0      0 192.168.1.150:8080      0.0.0.0:*               LISTEN      61029/haproxy
   tcp        0      0 192.168.1.150:80        0.0.0.0:*               LISTEN      61029/haproxy
   tcp        0      0 192.168.1.150:8977      0.0.0.0:*               LISTEN      61029/haproxy
   tcp        0      0 192.168.1.150:6080      0.0.0.0:*               LISTEN      61029/haproxy
   tcp        0      0 192.168.1.150:9696      0.0.0.0:*               LISTEN      61029/haproxy
   ```

```
tcp        0      0 192.168.1.150:8000      0.0.0.0:*               LISTEN      61029/haproxy
tcp        0      0 192.168.1.150:8004      0.0.0.0:*               LISTEN      61029/haproxy
tcp        0      0 192.168.1.150:8774      0.0.0.0:*               LISTEN      61029/haproxy
tcp        0      0 192.168.1.150:5000      0.0.0.0:*               LISTEN      61029/haproxy
tcp        0      0 192.168.1.150:8776      0.0.0.0:*               LISTEN      61029/haproxy
tcp        0      0 192.168.1.150:8041      0.0.0.0:*               LISTEN      61029/haproxy
```

注記

0.0.0.0のように、すべてのローカルアドレスをリッスンしているプロセスは、**192.168.1.150** からも利用できます。これらのプロセスには、**sshd**、**mysqld**、**dhclient**、**ntpd** などがあります。

ポート番号割り当ての表示

/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg ファイルを開き、デフォルトのポート番号割り当てを確認します。

以下の例は、ポート番号およびポートがリッスンするサービスを示しています。

- TCP ポート 6080: **nova_novncproxy**
- TCP ポート 9696: **neutron**
- TCP ポート 8000: **heat_cfn**
- TCP ポート 80: **horizon**
- TCP ポート 8776: **cinder**

この例では、**haproxy.cfg** ファイルで定義されているサービスの大半は、3 つすべてのコントローラーノードで **192.168.1.150** の IP アドレスをリッスンしています。ただし、**192.168.1.150** の IP アドレスを外部でリッスンしているのは **controller-0** ノードのみです。

このため、**controller-0** ノードで異常が発生した場合には、HAProxy は **192.168.1.150** を別のコントローラーノードに再割り当てするだけで、他のサービスはすべてフォールバックコントローラーノードですでに実行されている状態となります。

4.5. ペースメーカーのステータスと電源管理情報の表示

pcs status 出力の最後のセクションでは、IPMI などの電源管理フェンシングに関する情報と、Pacemaker サービス自体のステータスが表示されます。

```
 my-ipmilan-for-controller-0 (stonith:fence_ipmilan): Started my-ipmilan-for-controller-0
 my-ipmilan-for-controller-1 (stonith:fence_ipmilan): Started my-ipmilan-for-controller-1
 my-ipmilan-for-controller-2 (stonith:fence_ipmilan): Started my-ipmilan-for-controller-2

PCSD Status:
  overcloud-controller-0: Online
  overcloud-controller-1: Online
  overcloud-controller-2: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled openstack-cinder-volume       (systemd:openstack-cinder-volume):
```

```
Started overcloud-controller-0

  pcsd: active/enabled
```

my-ipmilan-for-controller の設定では、各コントローラーノードのフェンシングの種別 (**stonith:fence_ipmilan**) および IPMI サービスの稼働状態が分かります。PCSD Status は、3 つすべてコントローラーノードが現在オンラインであることを示します。また、Pacemaker サービスは **corosync**、**pacemaker**、および **pcsd** の 3 つのデーモンで構成されています。この例では、3 つすべてのサービスがアクティブかつ有効化されています。

4.6. 異常の発生したペースメーカーリソースのトラブルシューティング

Pacemaker リソースの 1 つで異常が発生した場合は、**pcs status** の出力に **Failed Actions** セクションを表示できます。以下の例では、**openstack-cinder-volume** サービスが **controller-0** で停止しています。

```
Failed Actions:
* openstack-cinder-volume_monitor_60000 on overcloud-controller-0 'not running' (7): call=74,
status=complete, exitreason='none',
 last-rc-change='Wed Dec 14 08:33:14 2016', queued=0ms, exec=0ms
```

このような場合には、systemd サービスの **openstack-cinder-volume** を有効にする必要があります。それ以外の場合には、問題を特定して修正し、続いてリソースをクリーンアップしなければならない場合があります。リソースの問題のトラブルシューティングに関する詳しい情報は、「8章 *リソースの問題のトラブルシューティング*」を参照してください。

第5章 STONITH を使用したコントローラノードのフェンシング

フェンシングとは、クラスターとクラスターリソースを保護するために、異常が発生したノードを分離するプロセスのことです。フェンシングがないと、異常が発生したノードが原因でクラスター内のデータが破損する可能性があります。

director は、Pacemaker を使用して、高可用性のコントローラーノードクラスターを提供します。Pacemaker は、異常が発生したノードをフェンシングするのに **STONITH** というプロセスを使用します。STONITH は、「Shoot the other node in the head」の略です。

コントローラーノードがヘルスチェックに失敗すると、Pacemaker 指定のコーディネーター (DC) として機能するコントローラーノードは、Pacemaker **stonith** サービスを使用して影響を受けるコントローラーノードをフェンシングします。

STONITH はデフォルトでは無効化されているため、Pacemaker がクラスター内の各ノードの電源管理を制御できるように手動で設定する必要があります。

重要

STONITH を使用しない高可用性オーバークラウドのデプロイはサポートの対象外です。高可用性オーバークラウドの Pacemaker クラスターの一部である各ノードには、STONITH デバイスを設定する必要があります。STONITH および Pacemaker の詳細は、「Fencing in a Red Hat High Availability Cluster」 および「Support Policies for RHEL High Availability Clusters - General Requirements for Fencing/STONITH」を参照してください。

Red Hat Enterprise Linux の Pacemaker を使用したフェンシングの詳細については、**『Red Hat Enterprise Linux 8 高可用性クラスターの設定および管理』**の「Red Hat High Availability クラスターでのフェンシングの設定」を参照してください。

5.1. サポート対象のフェンシングエージェント

フェンシング機能と共に高可用性環境をデプロイする場合、環境のニーズに応じて以下のフェンシングエージェントのいずれかを選択することができます。フェンシングエージェントを変更するには、「オーバークラウドでのフェンシングのデプロイとテスト」 で説明されているように、**fencing.yaml** ファイルで追加のパラメーターを設定する必要があります。

Intelligent Platform Management Interface (IPMI)

RHOSP がフェンシングの管理に使用するデフォルトのフェンシングメカニズム

Storage Block Device (SBD)

ウォッチドッグデバイスが設定されたデプロイメントで使用します。デプロイメントでは共有ストレージを使用しないでください。

fence_kdump

kdump クラッシュリカバリーサービスが設定されたデプロイメントで使用します。このエージェントを選択する場合、ダンプファイルを保存するのに十分なディスク容量を確保してください。このエージェントは、IPMI、**fence_rhevm**、または Redfish フェンシングエージェントに追加する、セカンダリーメカニズムとして設定することができます。複数のフェンシングエージェントを設定する場合は、2 番目のエージェントが次のタスクを開始する前に、1 番目のエージェントがタスクを完了するのに十分な時間を割り当てるようにしてください。

Redfish

DMTF Redfish API をサポートするサーバーが設定されたデプロイメントで使用します。このエージェントを指定するには、**fencing.yaml** ファイルで **agent** パラメーターの値を **fence_redfish** に変更します。Redfish についての詳細は、DTMF のドキュメント を参照してください。

oVirt および Red Hat Virtualization(RHV)のfence_rhevm

oVirt または RHV 環境で実行されるコントローラーノードのフェンシングを設定するのに使用します。IPMI フェンシングの場合と同じ様に **fencing.yaml** ファイルを生成することができますが、oVirt または RHV を使用するには、node **.json** ファイルで **pm_type** パラメーターを定義する必要があります。
デフォルトでは、**ssl_insecure** パラメーターは自己署名証明書を受け入れるように設定されます。セキュリティー要件に応じて、パラメーターの値を変更することができます。

5.2. オーバークラウドでのフェンシングのデプロイとテスト

フェンシングを設定するプロセスは、以下の段階で構成されます。

1. STONITH および Pacemaker の状態を確認する。
2. **fencing.yaml** ファイルを生成する。
3. オーバークラウドを再デプロイし設定をテストする。

前提条件

director でコントローラーノードを登録した際に作成した **nodes.json** ファイルにアクセスできるようにしてください。このファイルは、デプロイメント中に生成する **fencing.yaml** ファイルに必須なインプットになります。

STONITH および Pacemaker の状態の確認

1. それぞれのコントローラーノードに **heat-admin** ユーザーとしてログインします。
2. クラスターが動作状態にあることを確認します。

```
$ sudo pcs status
```

出力例:

```
Cluster name: openstackHA
Last updated: Wed Jun 24 12:40:27 2015
Last change: Wed Jun 24 11:36:18 2015
Stack: corosync
Current DC: lb-c1a2 (2) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
141 Resources configured
```

3. STONITH が無効になっていることを確認します。

```
$ sudo pcs property show
```

出力例:

```
Cluster Properties:
```

```
cluster-infrastructure: corosync
cluster-name: openstackHA
dc-version: 1.1.12-a14efad
have-watchdog: false
stonith-enabled: false
```

fencing.yaml 環境ファイルの生成

以下のいずれかのオプションを選択します。

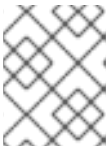
- IPMI または Red Hat Virtualization のフェンシングエージェントを使用する場合は、以下のコマンドを実行して **fencing.yaml** 環境ファイルを生成します。

```
$ openstack overcloud generate fencing --output fencing.yaml nodes.json
```

注記

- このコマンドにより、**ilo** および **drac** の電源管理情報が等価な IPMI 版に変換されます。
- **nodes.json** ファイルに、ノード上のいずれかのネットワークインターフェイス (NIC) の MAC アドレスが含まれていることを確認してください。詳しい情報は、「オーバークラウドノードの登録」を参照してください。

- Storage Block Device (SBD)、**fence_kdump**、または Redfish などの別のフェンシングエージェントを使用する場合は、**fencing.yaml** ファイルを手動で生成します。



注記

事前にプロビジョニングされたノードを使用する場合には、**fencing.yaml** ファイルも手動で作成する必要があります。

サポート対象のフェンシングエージェントについての詳しい情報は、「サポート対象のフェンシングエージェント」を参照してください。

オーバークラウドの再デプロイおよび設定のテスト

1. **overcloud deploy** コマンドを実行し、コントローラーノードでフェンシングを設定するために生成した **fencing.yaml** ファイルを含めます。

```
openstack overcloud deploy --templates \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
-e ~/templates/network-environment.yaml \
-e ~/templates/storage-environment.yaml --control-scale 3 --compute-scale 3 --ceph-storage-scale 3 --control-flavor control --compute-flavor Compute --ceph-storage-flavor ceph-storage --ntp-server pool.ntp.org --neutron-network-type vxlan --neutron-tunnel-types vxlan \
-e fencing.yaml
```

2. オーバークラウドにログインし、各コントローラーノードにフェンシングが設定されていることを確認します。

   a. Pacemaker がリソースマネージャーとして設定されていることを確認します。

```
$ source stackrc
```

```
$ nova list | grep controller
$ ssh heat-admin@<controller-x_ip>
$ sudo pcs status |grep fence
stonith-overcloud-controller-x (stonith:fence_ipmilan): Started overcloud-controller-y
```

この例では、Pacemaker は、**fencing.yaml** ファイルで指定された各コントローラーノードに STONITH リソースを使用するように設定されています。

注記

制御するのと同じノードに **fence-resource** プロセスを設定することはできません。

b. **pcs stonith show** コマンドを実行して、フェンシングリソースの属性を確認します。

```
$ sudo pcs stonith show <stonith-resource-controller-x>
```

STONITH 属性の値は、**fencing.yaml** ファイルの値と一致している必要があります。

コントローラーノードのフェンシングの確認

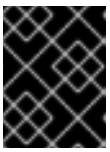
フェンシングが正しく機能するかどうかをテストするには、コントローラーノード上の全ポートを閉じ、サーバーを再起動してフェンシングをトリガーします。

1. コントローラーノードにログインします。

```
$ source stackrc
$ nova list |grep controller
$ ssh heat-admin@<controller-x_ip>
```

2. **root** ユーザーに変更し、各ポートで **iptables** コマンドを実行します。

```
$ sudo -i
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT &&
iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT &&
iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 5016 -j ACCEPT &&
iptables -A INPUT -p udp -m state --state NEW -m udp --dport 5016 -j ACCEPT &&
iptables -A INPUT ! -i lo -j REJECT --reject-with icmp-host-prohibited &&
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT &&
iptables -A OUTPUT -p tcp --sport 5016 -j ACCEPT &&
iptables -A OUTPUT -p udp --sport 5016 -j ACCEPT &&
iptables -A OUTPUT ! -o lo -j REJECT --reject-with icmp-host-prohibited
```



重要

このステップによりコントローラーノードへの接続がすべて切断されるので、サーバーがリブートします。

3. 別のコントローラーノードから、Pacemaker のログファイルでフェンシングイベントの有無を確認します。

```
$ ssh heat-admin@<controller-x_ip>
$ less /var/log/cluster/corosync.log
(less): /fenc*
```

- STONITH サービスがコントローラーでフェンシングアクションを実行していれば、ログファイルにフェンシングイベントが記録されます。

4. 数分待ってから、**pcs status** コマンドを実行して、再起動したコントローラーノードがクラスターで再び実行されていることを確認します。

5.3. STONITH 情報の表示

STONITH がフェンシングデバイスをどのように設定するかを確認するには、オーバークラウドから **pcs stonith show --full** コマンドを実行します。

```
$ sudo pcs stonith show --full
 Resource: my-ipmilan-for-controller-0 (class=stonith type=fence_ipmilan) 1
  Attributes: pcmk_host_list=overcloud-controller-0 ipaddr=10.100.0.51 login=admin passwd=abc
lanplus=1 cipher=3
  Operations: monitor interval=60s (my-ipmilan-for-controller-0-monitor-interval-60s)
 Resource: my-ipmilan-for-controller-1 (class=stonith type=fence_ipmilan)
  Attributes: pcmk_host_list=overcloud-controller-1 ipaddr=10.100.0.52 login=admin passwd=abc
lanplus=1 cipher=3
  Operations: monitor interval=60s (my-ipmilan-for-controller-1-monitor-interval-60s)
 Resource: my-ipmilan-for-controller-2 (class=stonith type=fence_ipmilan)
  Attributes: pcmk_host_list=overcloud-controller-2 ipaddr=10.100.0.53 login=admin passwd=abc
lanplus=1 cipher=3
  Operations: monitor interval=60s (my-ipmilan-for-controller-2-monitor-interval-60s)
```

--full オプションは、3 つのコントローラーノードに関するフェンシングの詳細を返します。

この出力には、リソースごとに以下の情報が表示されます。

- フェンシングデバイスが必要に応じてマシンの電源をオン/オフするのに使用する IPMI 電源管理サービス (例: **fence_ipmilan**)
- IPMI インターフェースの IP アドレス (例: **10.100.0.51**)
- ログインに使用するユーザー名 (例: **admin**)
- ノードにログインするのに使用するパスワード (例: **abc**)
- それぞれのホストをモニターする間隔 (例: **60** 秒)

5.4. フェンシングパラメーター

オーバークラウドにフェンシングをデプロイする際には、フェンシングを設定するのに必要なパラメーターを定義して **fencing.yaml** ファイルを生成します。フェンシングのデプロイおよびテストについての詳細は、「オーバークラウドでのフェンシングのデプロイとテスト」を参照してください。

fencing.yaml 環境ファイルの構成例を以下に示します。

```
parameter_defaults:
  EnableFencing: true
  FencingConfig:
    devices:
    - agent: fence_ipmilan
      host_mac: 11:11:11:11:11:11
```

```
    params:
      ipaddr: 10.0.0.101
      lanplus: true
      login: admin
      passwd: InsertComplexPasswordHere
      pcmk_host_list: host04
      privlvl: administrator
```

このファイルには以下のパラメーターが含まれます。

EnableFencing

Pacemaker の管理するノードのフェンシング機能を有効にします。

FencingConfig

フェンシングデバイスおよび各デバイスのパラメーターを一覧表示します。

- **agent**: フェンシングエージェント名。Red Hat OpenStack Platform でサポートされるのは、IPMI 用の **fence_ipmilan** だけです。
- **host_mac**: フェンスデバイスの一意の ID。
- **params**: フェンスデバイスパラメーターの一覧。

フェンシングデバイスのパラメーター

- **auth**: IPMI 認証種別 (**md5**、**password**、または none)
- **ipaddr**: IPMI IP アドレス
- **ipport**: IPMI ポート
- **login**: IPMI デバイス用のユーザー名
- **passwd**: IPMI デバイス用のパスワード
- **lanplus**: lanplus を使用して、接続のセキュリティーを向上させます。
- **privlvl**: IPMI デバイスの権限レベル
- **pcmk_host_list**: Pacemaker ホストの一覧

第6章 HAPROXY を使用したトラフィック負荷の分散

HAProxy サービスは、トラフィックの負荷を高可用性クラスター内のコントローラーノードに分散する機能に加えて、ロギングおよびサンプル設定を提供します。

haproxy パッケージに含まれる **haproxy** デーモンは、同じ名前の **systemd** サービスに対応します。Pacemaker は、HAProxy サービスを **haproxy-bundle** と呼ばれる高可用性サービスとして管理します。

HAProxy についての詳細は、HAProxy 1.8 のドキュメント を参照してください。

HAProxy が正しく設定されていることを確認する方法は、KCS のアーティクル「haproxy.cfg が OpenStack のサービスをロードバランシングできるよう正しく設定されているかどうか、確認する方法はありますか?」を参照してください。

6.1. HAPROXY の仕組み

director は、ほとんどの Red Hat OpenStack Platform サービスを HAProxy サービスを使用するように設定することができます。Director は、これらのサービスを **/var/lib/config-data/haproxy/etc/haproxy/haproxy.cfg** ファイルで設定します。このファイルは、HAProxy が各オーバークラウドノードの専用のコンテナーで実行されるように指示します。

HAProxy が管理するサービスの一覧を以下の表に示します。

表6.1 HAProxy が管理するサービス

aodh	cinder	glance_api	gnocchi
haproxy.stats	heat_api	heat_cfn	horizon
keystone_admin	keystone_public	mysql	neutron
nova_metadata	nova_novncproxy	nova_osapi	nova_placement

haproxy.cfg ファイル内の各サービスで、以下の属性が設定されます。

- **listen**: 要求をリッスンするサービスの名前
- **bind**: サービスがリッスンする IP アドレスおよび TCP ポート番号
- **server**: HAProxy を使用する各コントローラーノードサーバーの名前、IP アドレスおよびリッスンするポート、ならびにサーバーに関する追加情報

以下の例は、**haproxy.cfg** ファイル内の OpenStack Block Storage (cinder) サービスの設定を示しています。

```
listen cinder
  bind 172.16.0.10:8776
  bind 192.168.1.150:8776
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
```

```
server overcloud-controller-0 172.16.0.13:8777 check fall 5 inter 2000 rise 2
server overcloud-controller-1 172.16.0.14:8777 check fall 5 inter 2000 rise 2
server overcloud-controller-2 172.16.0.15:8777 check fall 5 inter 2000 rise 2
```

この出力例は、OpenStack Block Storage (cinder) サービスに関する以下の情報を示しています。

- **172.16.0.10:8776**: オーバークラウド内で使用する内部 API ネットワーク (VLAN201) 上の仮想 IP アドレスおよびポート
- **192.168.1.150:8776**:オーバークラウド外から API ネットワークへのアクセスを提供する外部ネットワーク (VLAN100) 上の仮想 IP アドレスおよびポート
- **8777**: OpenStack Block Storage(cinder)サービスがリッスンしているポート番号。
- **server**: コントローラーノード名および IP アドレス。HAProxy は、これらの IP アドレスに送信された要求を **server** の出力に一覧表示されるコントローラーノードのいずれかに転送することができます。
- **httpchk**: コントローラーノードサーバーでのヘルスチェックを有効にします。
- **fall 5**: サービスがオフラインであると判断されるヘルスチェックの失敗回数
- **inter 2000**: 連続する 2 回のヘルスチェックの間隔 (ミリ秒単位)
- **rise 2**: サービスが動作中であると判断されるヘルスチェックの成功回数

haproxy.cfg ファイルで使用できる設定の詳細は、**haproxy** パッケージがインストールされている任意のノードの **/usr/share/doc/haproxy-[VERSION]/configuration.txt** ファイルを参照してください。

6.2. HAPROXY 統計の表示

すべての HA デプロイメントにおいて、director によりデフォルトで HAProxy Stats (統計) も有効になります。この機能により、データ転送、接続、およびサーバーの状態についての詳細情報を HAProxy Stats のページで確認することができます。

director は、HAProxy Stats ページにアクセスするのに使用する **IP:Port** アドレスも設定し、その情報を **haproxy.cfg** ファイルに保存します。

手順

1. HAProxy がインストールされている任意のコントローラーノードで **/var/lib/config-data/haproxy/etc/haproxy/haproxy.cfg** ファイルを開きます。
2. **listen haproxy.stats** セクションを探します。

```
listen haproxy.stats
  bind 10.200.0.6:1993
  mode http
  stats enable
  stats uri /
  stats auth admin:<haproxy-stats-password>
```

3. Web ブラウザーで **10.200.0.6:1993** に移動し、**stats auth** 行の認証情報を入力して HAProxy Stats ページを表示します。

第7章 GALERA を使用したデータベースレプリケーションの管理

Red Hat OpenStack Platform では、データベースレプリケーションの管理に MariaDB Galera Cluster を使用します。Pacemaker は、データベースのマスター/スレーブのステータスを管理する **バンドルセット** リソースとして、Galera サービスを実行します。Galera を使用して、ホスト名の解決、クラスターの整合性、ノードの整合性、データベースレプリケーションのパフォーマンス等、データベースクラスターのさまざまな要素をテストおよび検証することができます。

他の Pacemaker サービスと同様に、**pcs status** コマンドを使用して、Galera サービスが実行されていることと、それが実行されているコントローラーノードを確認できます。Pacemaker バンドルステータスの表示についての詳細は、「バンドルのステータスの確認」を参照してください。

データベースクラスターの整合性を詳しく調べる際には、各ノードは以下の条件を満たしている必要があります。

- ノードが正しいクラスターの一部である。
- ノードがクラスターに書き込み可能である。
- ノードがクラスターからのクエリーおよび書き込みコマンドを受け取ることができる。
- ノードがクラスター内の他のノードに接続されている。
- ノードが Write Set をローカルデータベースのテーブルにレプリケーションしている。

7.1. ホスト名の解決の確認

デフォルトでは、director は Galera リソースを IP アドレスにではなくホスト名にバインドします。そのため、ホスト名の解決を妨げる問題 (例: DNS の設定不良や異常など) が原因で、Pacemaker による Galera リソースの管理が不適切になる場合があります。

MariaDB Galera クラスターのトラブルシューティングを行うには、まずホスト名解決の問題を取り除き、続いて各コントローラーノードのデータベースで Write Set のレプリケーションステータスを確認します。MySQL データベースにアクセスするには、オーバークラウドのデプロイメント時に director が設定したパスワードを使用します。

手順

1. コントローラーノードから **hiera** コマンドを実行し、MariaDB データベースの root パスワードを取得します。

```
$ sudo hiera -c /etc/puppet/hiera.yaml "mysql::server::root_password"
*[MYSQL-HIERA-PASSWORD]*
```

2. ノードで実行される MariaDB コンテナーの名前を取得します。

```
$ sudo podman ps | grep -i galera
a403d96c5026  undercloud.ctlplane.localdomain:8787/rhosp-rhel8/openstack-mariadb:16.0-
106           /bin/bash /usr/lo...  3 hours ago  Up 3 hours ago         galera-bundle-podman-0
```

3. 各ノードの MariaDB データベースから、Write Set のレプリケーション情報を取得します。

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-
PASSWORD]" -e "SHOW GLOBAL STATUS LIKE 'wsrep_%';"
    +----------------------------+----------+
```

```
| Variable_name              | Value    |
+----------------------------+----------+
| wsrep_applier_thread_count | 1        |
| wsrep_apply_oooe           | 0.018672 |
| wsrep_apply_oool           | 0.000630 |
| wsrep_apply_window         | 1.021942 |
| ...                        | ...      |
+----------------------------+----------+
```

関連する変数はそれぞれ **wsrep** のプレフィックスを使用します。

4. まず初めにクラスターが正しいノード数を報告することを確認して、MariaDB Galera Cluster の健全性および整合性を検証します。

7.2. データベースクラスターの整合性の確認

MariaDB Galera クラスターの問題を調査する場合、各コントローラーノードで特定の **wsrep** データベース変数をチェックすることで、クラスター全体の整合性をチェックできます。

手順

次のコマンドを実行し、**VARIABLE** を確認する **wsrep** データベース変数に置き換えます。

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW GLOBAL STATUS LIKE '**VARIABLE**';"
```

以下の例で、ノードのクラスター状態の UUID を表示する方法を説明します。

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW GLOBAL STATUS LIKE 'wsrep_cluster_state_uuid';"

+--------------------------+--------------------------------------+
| Variable_name            | Value                                |
+--------------------------+--------------------------------------+
| wsrep_cluster_state_uuid | e2c9a15e-5485-11e0-0800-6bbb637e7211 |
+--------------------------+--------------------------------------+
```

以下の表には、クラスターの整合性を確認するのに使用できる **wsrep** データベース変数をまとめています。

表7.1 クラスターの整合性を確認するためのデータベース変数

変数	概要	説明
wsrep_cluster_state_uuid	クラスターの状態の UUID	ノードが属するクラスターの ID。全ノードに同一のクラスター ID が割り当てられている必要があります。異なる ID が割り当てられたノードは、クラスターに接続できません。

変数	概要	説明
wsrep_cluster_size	クラスター内のノード数	任意のノードで確認することができます。値が実際のノード数を下回る場合には、いずれかのノードで異常が発生したか、接続が失われたことを意味します。
wsrep_cluster_conf_id	クラスターの全変更回数	クラスターが複数のコンポーネント (**パーティション** とも呼ばれる) に分割されたかどうかを判断します。通常、ネットワーク障害が原因でパーティションが発生します。全ノードで値が同一でなければなりません。 異なる **wsrep_cluster_conf_id** を報告するノードがある場合には、**wsrep_cluster_status** の値をチェックして、ノードがクラスターにまだ書き込み可能かどうかを確認します(**Primary**)。
wsrep_cluster_status	Primary コンポーネントのステータス	ノードがクラスターに書き込み可能かどうかを判断します。ノードがクラスターに書き込み可能であれば、**wsrep_cluster_status** の値は **Primary** になります。それ以外の値の場合には、ノードが稼働していないパーティションの一部であることを示します。

7.3. データベースノードの整合性の確認

Galera クラスターの問題を特定のノードに切り分けることができれば、特定の **wsrep** データベース変数がそのノードの特定の問題を示している可能性があります。

手順

次のコマンドを実行し、**VARIABLE** を確認する **wsrep** データベース変数に置き換えます。

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW GLOBAL STATUS LIKE 'VARIABLE';"
```

以下の表には、ノードの整合性を確認するのに使用できる **wsrep** データベース変数をまとめています。

表7.2 ノードの整合性を確認するためのデータベース変数

変数	概要	説明
wsrep_ready	ノードがクエリーを受け入れる能力	ノードがクラスターから Write Set を受け入れることができるかどうかを示します。その場合には、**wsrep_ready** は **ON** になります。
wsrep_connected	ノードのネットワーク接続性	ノードがネットワーク上の他のノードに接続できるかどうかを示します。その場合には、**wsrep_connected** は **ON** になります。
wsrep_local_state_comment	ノードの状態	ノードの状態の概要を示します。ノードがクラスターに書き込み可能であれば、**wsrep_local_state_comment** の一般的な値は **Joining**、**Waiting on SST**、**Joined**、**Synced**、または **Donor** です。 ノードが稼働していないコンポーネントの一部の場合には、**wsrep_local_state_comment** の値は **Initialized** になります。

注記

- ノードがクラスター内のノードのサブセットにしか接続されている場合でも、**wsrep_connected** の値は **ON** になります。たとえば、クラスターのパーティションの場合には、そのノードは、クラスターに書き込みができないコンポーネントの一部となっている可能性があります。クラスターの整合性の確認に関する詳細は、「データベースクラスターの整合性の確認」を参照してください。
- **wsrep_connected** 値が **OFF** の場合、ノードはどのクラスターコンポーネントにも接続されていません。

7.4. データベースレプリケーションのパフォーマンスのテスト

クラスターおよび個々のノードがすべて健全で安定している場合、特定のデータベース変数のクエリーを行い、レプリケーションのスループットについてパフォーマンスのベンチマークテストを行うことができます。

これらの変数の 1 つのクエリーを行うたびに、**FLUSH STATUS** コマンドは変数の値をリセットします。ベンチマークテストを行うには、複数のクエリーを実行し、差異を分析する必要があります。この差異は、Flow Control がクラスターのパフォーマンスにどの程度影響を及ぼしているかを判断するのに役立ちます。

Flow Control は、クラスターがレプリケーションを制御するのに使用するメカニズムです。ローカルの **受信キュー** が一定のしきい値を超えると、キューのサイズが減少するまで、Flow Control はレプリケーションを一時停止します。Flow Control の詳細は、Galera Cluster の Web サイトで「Flow Control」を参照してください。

手順

次のコマンドを実行し、**VARIABLE** を確認する **wsrep** データベース変数に置き換えます。

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW STATUS LIKE '%VARIABLE%';"
```

以下の表には、データベースレプリケーションのパフォーマンスをテストするのに使用できるさまざまな **wsrep** データベース変数をまとめています。

表7.3 データベースレプリケーションのパフォーマンスを確認するためのデータベース変数

変数	概要	用途
wsrep_local_recv_queue_avg	最後のクエリー後のローカル受信 Write Set キューの平均サイズ	値が 0.0 を超えていれば、ノードが受信速度に対応して Write Set を適用できないことが分かります。この場合、レプリケーションのスロットリングがトリガーされます。このベンチマークの詳しい情報は、**wsrep_local_recv_queue_min** と **wsrep_local_recv_queue_max** を確認してください。
wsrep_local_send_queue_avg	最後のクエリー後の送信キューの平均長さ	値が 0.0 を超えていれば、レプリケーションのスロットルおよびネットワークスループットの問題の可能性が高いことが分かります。
wsrep_local_recv_queue_min および **wsrep_local_recv_queue_max**	最後のクエリー後のローカル受信キューの最小/最大サイズ	**wsrep_local_recv_queue_avg** の値が 0.0 を超える場合、これらの変数を確認してキューサイズの範囲を判断することができます。

変数	概要	用途
wsrep_flow_control_paused	最後のクエリー以降、Flow Control がノードを一時停止した時間の割合	値が 0.0 を超えていれば、Flow Control がノードを一時停止したことが分かります。一時停止の時間を把握するには、**wsrep_flow_control_paused** の値をクエリー間の秒数で乗算します。可能な限り 0.0 に近い値が最適値です。 以下に例を示します。 ● 最後のクエリーから 1 分後の **wsrep_flow_control_paused** の値が 0.50 の場合、Flow Control は 30 秒間ノードを一時停止しています。 ● 最後のクエリーから 1 分後に **wsrep_flow_control_paused** の値が 1.0 の場合、Flow Control はその 1 分間にわたってノードを一時停止します。
wsrep_cert_deps_distance	並行して適用することのできるシーケンス番号 (**seqno**) の最小値と最大値の差の平均	スロットリングおよび一時停止の場合、この変数は並行して適用することのできる Write Set 数の平均を示します。この値を **wsrep_slave_threads** 変数と比較して、実際に同時に適用することのできる Write Set の数を確認します。

変数	概要	用途
wsrep_slave_threads	同時に適用することのできるスレッドの数	この変数の値を増やして、より多くのスレッドを同時に適用することができます。これにより、**wsrep_cert_deps_distance** の値も増加します。**wsrep_slave_threads** の値は、ノードの CPU コア数よりも大きくすることはできません。 たとえば、**wsrep_cert_deps_distance** の値が **20** の場合は、**wsrep_slave_threads** の値を **2** から **4** の値を増やして、ノードを適用することができる write set の量を増やすことができます。 問題のあるノードの **wsrep_slave_threads** の値が既に最適値である場合、接続性の問題を調査する際に、ノードをクラスターから除外することができます。

第8章 リソースの問題のトラブルシューティング

リソースに異常が発生した場合は、問題の原因と場所を調査し、異常が発生したリソースを修正し、必要に応じてリソースをクリーンアップする必要があります。デプロイメントによって、リソース異常にはさまざまな原因が考えられ、リソースを調査して問題の修正方法を決定する必要があります。

たとえば、リソースの制約を確認することで、リソースが相互に障害とならないようにし、また互いに接続できるようにすることができます。他のコントローラーノードよりも頻繁にフェンシングされるコントローラーノードを調査し、通信の問題を識別できる場合もあります。

8.1. リソースの制約の表示

サービスがどのように起動されるかに関する制約を表示することができます。これには、各リソースが配置される場所、リソースが起動される順序、別のリソースと共に配置する必要があるかどうか、などの制約が含まれます。

すべてのリソース制約の表示

任意のコントローラーノードで、**pcs constraint show** コマンドを実行します。

```
$ sudo pcs constraint show
```

以下の例には、Controller ノードで **pcs constraint show** コマンドを実行した場合の出力を示しており、途中省略しています。

```
Location Constraints:
  Resource: galera-bundle
    Constraint: location-galera-bundle (resource-discovery=exclusive)
      Rule: score=0
        Expression: galera-role eq true
  [...]
  Resource: ip-192.168.24.15
    Constraint: location-ip-192.168.24.15 (resource-discovery=exclusive)
      Rule: score=0
        Expression: haproxy-role eq true
  [...]
  Resource: my-ipmilan-for-controller-0
    Disabled on: overcloud-controller-0 (score:-INFINITY)
  Resource: my-ipmilan-for-controller-1
    Disabled on: overcloud-controller-1 (score:-INFINITY)
  Resource: my-ipmilan-for-controller-2
    Disabled on: overcloud-controller-2 (score:-INFINITY)
Ordering Constraints:
  start ip-172.16.0.10 then start haproxy-bundle (kind:Optional)
  start ip-10.200.0.6 then start haproxy-bundle (kind:Optional)
  start ip-172.19.0.10 then start haproxy-bundle (kind:Optional)
  start ip-192.168.1.150 then start haproxy-bundle (kind:Optional)
  start ip-172.16.0.11 then start haproxy-bundle (kind:Optional)
  start ip-172.18.0.10 then start haproxy-bundle (kind:Optional)
Colocation Constraints:
  ip-172.16.0.10 with haproxy-bundle (score:INFINITY)
  ip-172.18.0.10 with haproxy-bundle (score:INFINITY)
  ip-10.200.0.6 with haproxy-bundle (score:INFINITY)
```

```
  ip-172.19.0.10 with haproxy-bundle (score:INFINITY)
  ip-172.16.0.11 with haproxy-bundle (score:INFINITY)
  ip-192.168.1.150 with haproxy-bundle (score:INFINITY)
```

この出力には、以下の主要な制約種別が表示されています。

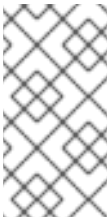
Location Constraints

リソースを割り当てることのできる場所を一覧表示します。

- 最初の制約は、**galera-role** 属性が **true** に設定されたノードで実行する **galera-bundle** リソースを設定するルールを定義します。
- 場所に関する 2 番目の制約は、IP リソース **ip-192.168.24.15** は **haproxy-role** 属性が **true** に設定されたノードでのみ実行されることを指定します。これは、クラスターが IP アドレスを **haproxy** サービスに関連付けることを意味し、サービスを到達可能にするために必要です。
- 場所に関する 3 番目の制約は、**ipmilan** リソースが各コントローラーノードで無効化されることを意味します。

Ordering Constraints

リソースを起動することのできる順序を一覧表示します。この例は、仮想 IP アドレスリソース **IPaddr2** は HAProxy サービスより先に起動しなければならない、という制約を示しています。



注記

順序に関する制約は、IP アドレスリソースおよび HAProxy にのみ適用されます。Compute などのサービスは、Galera などの依存関係にあるサービスの中断に対する耐性があると予想されるため、その他すべてのリソースは systemd により管理されます。

Colocation Constraints

共に配置する必要があるリソースを一覧表示します。すべての仮想 IP アドレスは **haproxy-bundle** リソースにリンクされています。

Galera の場所に関する制約の表示

任意のコントローラーノードで、**pcs property show** コマンドを実行します。

```
$ sudo pcs property show
```

出力例:

```
Cluster Properties:
 cluster-infrastructure: corosync
 cluster-name: tripleo_cluster
 dc-version: 2.0.1-4.el8-0eb7991564
 have-watchdog: false
 redis_REPL_INFO: overcloud-controller-0
 stonith-enabled: false
Node Attributes:
 overcloud-controller-0: cinder-volume-role=true galera-role=true haproxy-role=true rabbitmq-
role=true redis-role=true rmq-node-attr-last-known-rabbitmq=rabbit@overcloud-controller-0
 overcloud-controller-1: cinder-volume-role=true galera-role=true haproxy-role=true rabbitmq-
```

```
role=true redis-role=true rmq-node-attr-last-known-rabbitmq=rabbit@overcloud-controller-1
 overcloud-controller-2: cinder-volume-role=true galera-role=true haproxy-role=true rabbitmq-
role=true redis-role=true rmq-node-attr-last-known-rabbitmq=rabbit@overcloud-controller-2
```

この出力では、**galera-role** 属性がすべてのコントローラーノードで **true** であることを確認できます。これは、**galera-bundle** リソースがこれらのノードでのみ実行されることを意味します。これと同じ概念が、場所に関する他の制約に関連付けられたその他の属性に適用されます。

8.2. コントローラーノードのリソースに関する問題の調査

問題の種別や場所に応じて、リソースを調査して修正するのにさまざまなアプローチを用いることができます。

コントローラーノードに関する問題の調査

コントローラーノードのヘルスチェックに失敗した場合は、コントローラーノード間の通信に問題があることを示しています。問題を調査するには、コントローラーノードにログインして、サービスが正常に起動できるかどうかを確認します。

個別のリソースに関する問題の調査

Controller 上のほとんどのサービスが正常に実行されている場合は、**pcs status** コマンドを実行して、出力で特定のサービス障害に関する情報を確認できます。また、リソースに異常が発生しているコントローラーにログインして、コントローラーノードのリソースの動作を調べることもできます。

手順

リソース以下の手順では、**openstack-cinder-volume** リソースを調査する方法を説明します。

1. リソースに異常が発生しているコントローラーノードを特定し、ログインします。
2. **systemctl status** コマンドを実行し、リソースのステータスおよび最近のログイベントを表示します。

```
[heat-admin@overcloud-controller-0 ~]$ sudo systemctl status openstack-cinder-volume
● openstack-cinder-volume.service - Cluster Controlled openstack-cinder-volume
   Loaded: loaded (/usr/lib/systemd/system/openstack-cinder-volume.service; disabled;
vendor preset: disabled)
  Drop-In: /run/systemd/system/openstack-cinder-volume.service.d
           └─50-pacemaker.conf
   Active: active (running) since Tue 2016-11-22 09:25:53 UTC; 2 weeks 6 days ago
 Main PID: 383912 (cinder-volume)
   CGroup: /system.slice/openstack-cinder-volume.service
           ├─383912 /usr/bin/python3 /usr/bin/cinder-volume --config-file /usr/share/cinder/cinder-
dist.conf --config-file /etc/cinder/cinder.conf --logfile /var/log/cinder/volume.log
           └─383985 /usr/bin/python3 /usr/bin/cinder-volume --config-file /usr/share/cinder/cinder-
dist.conf --config-file /etc/cinder/cinder.conf --logfile /var/log/cinder/volume.log


Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:55.798 383912 WARNING oslo_config.cfg [req-8f32db96-7ca2-4fc5-82ab-
271993b28174 - - - -...e future.
Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:55.799 383912 WARNING oslo_config.cfg [req-8f32db96-7ca2-4fc5-82ab-
271993b28174 - - - -...e future.
Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:55.926 383985 INFO cinder.coordination [-] Coordination backend started
```

```
successfully.
Nov 22 09:25:55 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:55.926 383985 INFO cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - ...r (1.2.0)
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.047 383985 WARNING oslo_config.cfg [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - - -...e future.
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.048 383985 WARNING oslo_config.cfg [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - - -...e future.
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.048 383985 WARNING oslo_config.cfg [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - - -...e future.
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.063 383985 INFO cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - ...essfully.
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.111 383985 INFO cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - ...r (1.2.0)
Nov 22 09:25:56 overcloud-controller-0.localdomain cinder-volume[383912]: 2016-11-22
09:25:56.146 383985 INFO cinder.volume.manager [req-cb07b35c-af01-4c45-96f1-
3d2bfc98ecb5 - - ...essfully.
Hint: Some lines were ellipsized, use -l to show in full.
```

3. 出力からの情報に基づいて、異常が発生したリソースを修正します。

4. **pcs resource cleanup** コマンドを実行し、リソースのステータスおよび異常回数をリセットします。

```
$ sudo pcs resource cleanup openstack-cinder-volume
  Resource: openstack-cinder-volume successfully cleaned up
```

第9章 高可用性 RED HAT CEPH STORAGE クラスターのモニタリング

Red Hat Ceph Storage と共にオーバークラウドをデプロイする場合、Red Hat OpenStack Platform は **ceph-mon** モニターデーモンを使用して Ceph クラスターを管理します。director はデーモンをすべてのコントローラーノードにデプロイします。

Ceph Monitoring サービスのステータスの表示

コントローラーノードで **service ceph status** コマンドを実行して、Ceph Monitoring サービスが実行中であることを確認します。

```
$ sudo service ceph status
=== mon.overcloud-controller-0 ===
mon.overcloud-controller-0: running {"version":"0.94.1"}
```

Ceph Monitoring 設定の表示

コントローラーノードまたは Ceph ノードで、**/etc/ceph/ceph.conf** ファイルを開いてモニタリング設定パラメーターを表示します。

```
[global]
osd_pool_default_pgp_num = 128
osd_pool_default_min_size = 1
auth_service_required = cephx
mon_initial_members = overcloud-controller-0,overcloud-controller-1,overcloud-controller-2
fsid = 8c835acc-6838-11e5-bb96-2cc260178a92
cluster_network = 172.19.0.11/24
auth_supported = cephx
auth_cluster_required = cephx
mon_host = 172.18.0.17,172.18.0.15,172.18.0.16
auth_client_required = cephx
osd_pool_default_size = 3
osd_pool_default_pg_num = 128
public_network = 172.18.0.17/24
```

この例は、以下の情報を示しています。

- **mon_initial_members** パラメーターにより、3 つすべてのコントローラーノードは、Red Hat Ceph Storage クラスターをモニターするように設定されています。
- コントローラーノードと Red Hat Ceph Storage ノード間の通信パスを提供するために、**172.19.0.11/24** ネットワークが設定されています。
- Red Hat Ceph Storage ノードはコントローラーノードとは別のネットワークに割り当てられ、モニタリングするコントローラーノードの IP アドレスは **172.18.0.15**、**172.18.0.16**、および **172.18.0.17** です。

個別の Ceph ノードのステータスの表示

Ceph ノードにログインし、**ceph -s** コマンドを実行します。

```
# ceph -s
    cluster 8c835acc-6838-11e5-bb96-2cc260178a92
     health HEALTH_OK
```

```
    monmap e1: 3 mons at {overcloud-controller-0=172.18.0.17:6789/0,overcloud-controller-
1=172.18.0.15:6789/0,overcloud-controller-2=172.18.0.16:6789/0}
           election epoch 152, quorum 0,1,2 overcloud-controller-1,overcloud-controller-2,overcloud-
controller-0
    osdmap e543: 6 osds: 6 up, 6 in
     pgmap v1736: 256 pgs, 4 pools, 0 bytes data, 0 objects
          267 MB used, 119 GB / 119 GB avail
               256 active+clean
```

この出力例からは、**health** パラメーターの値が **HEALTH_OK** であることが分かります。これは、Ceph ノードがアクティブで正常であることを示します。この出力には、3 つの **overcloud-controller** ノード上で実行されている 3 つの Ceph Monitoring サービス、ならびにこれらのサービスの IP アドレスおよびポートも示されています。

Red Hat Ceph Storage についての詳しい情報は、Red Hat Ceph Storage の製品ページ を参照してください。